Backward Tree Pattern Matching

J. Trávníček

Faculty of Information Technologies
Czech Technical University in Prague

MELA 2012
28. 9. 2012

Outline

1. Introduction
   - Motivation
   - Basic Notions
2. Tree Matching
   - Problem Definition
   - Backward subtree matching
   - Backward tree pattern matching

Outline

Motivation

- Arbology applies well known principles of string pattern matching to processing of trees in linear notation.
- Backward pattern matching (Boyer-Moore algorithm or Horspool algorithm) in strings proved to be efficient for various applications.
- Tree in linear notation can be seen as string.

Outline

1. Introduction
   - Motivation
   - Basic Notions
2. Tree Matching
   - Problem Definition
   - Backward subtree matching
   - Backward tree pattern matching

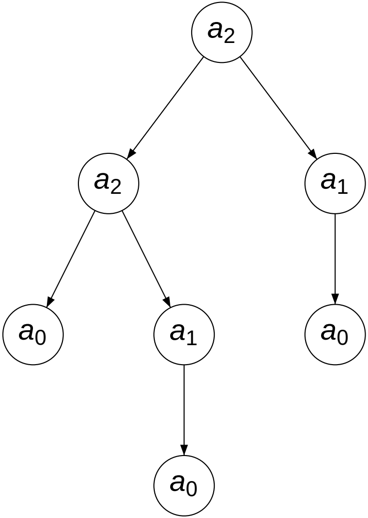
Subject Tree

- Ranked alphabet
  $\mathcal{A} = \{a_2, a_1, a_0\}$
- Unranked alphabet
  $\mathcal{A} = \{a, |\}$
- Subject tree t in prefix bar notation

$$pref\_bar(t) = a\,a\,a \mid a\,a \mid \mid \mid a\,a \mid \mid \mid$$

- Subject tree t in prefix notation

$$pref(t) = a_2\, a_2\, a_0\, a_1\, a_0\, a_1\, a_0$$



Arity checksum

- Arity checksum for trees in ranked alphabet:
  $ac(pref(t)) = arity(a_1) + arity(a_2) + \ldots + arity(a_m) - m + 1$
  $= \sum_{i=1}^{m} arity(a_i) - m + 1$
- Arity checksum for bar notation:
  $ac(pref\_bar(t)) = |pref\_bar(t)|_a - |pref\_bar(t)|_|$

Tree properties

- Trees in linear notation have arity checksum equal to zero.
- Trees in linear notation have only trivial borders.

Outline

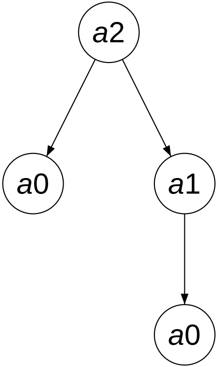
Subtree

- Alphabet of tree pattern

$$\mathcal{A} = \{a_2, a_1, a_0\}$$

- Tree pattern p_1 in prefix notation

$$pref(p_1) = a_2\ a_0\ a_1\ a_0$$



Tree Pattern

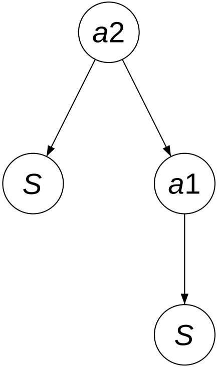
- Alphabet of tree pattern

$$\mathcal{A} = \{a_2, a_1, a_0, S\}$$

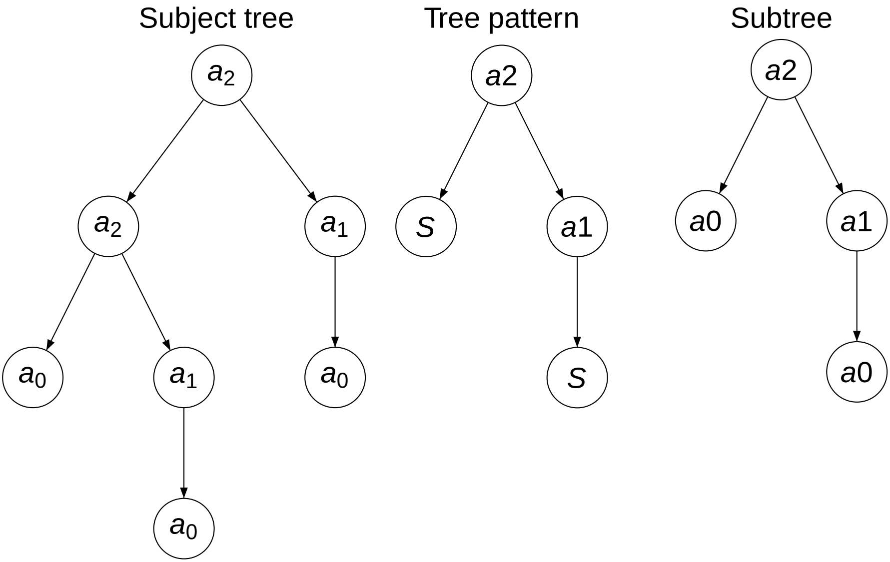
- Tree pattern p_2 in prefix notation

$$pref(p_2) = a_2\ S\ a_1\ S$$

- S is a linear variable.



Tree Pattern (Subtree)



Outline

Bad character shift

- Bad character shift makes use of bad character shift table.
- Length of the maximal safe shift is stored for each symbol.

Alphabet $\mathcal{A} = \{a_3, a_2, a_1, a_0\}$, subtree $pref(p_1) = a_2\ a_0\ a_1\ a_0$.

Table: Bad character shift table

a_3	a_2	a_1	a_0
4	3	1	2

Good suffix shift

- Good suffix shift makes use of good suffix shift table.
- Length of the maximal safe shift is stored for the number of sucessfully compared symbols.
- Length of the maximal safe shift is limited by the border of the pattern.

Subtree $pref(p_1) = a_2\ a_0\ a_1\ a_0$.

Table: Good suffix shift table

4	3	2	1	0
4	4	4	2	1

Other principles

- Backward dawg matching
- Backward factor matching
- Backward oracle matching

Outline

Related issues

Subject tree
$pref(t) = a_2\ a_2\ a_0\ a_1\ a_0\ a_1\ a_0$
$pref\_bar(t) = a\ a\ a \mid a\ a \mid \mid \mid a\ a \mid \mid \mid$

Tree pattern
$pref(p_2) = a_2\ S\ a_1\ S$
$pref\_bar(p_2) = a\ S \mid a\ S \mid \mid \mid$

- Subtree variable S is matched to more symbols.
- Symbols matched to the subtree variable are "unknown".

Bad character shift

- Again, length of the maximal safe shift is stored for each symbol.
- Both bar and ranked alphabet provide some usefull information – combination of both can be used.

Alphabet $\mathcal{A} = \{a_3, a_2, a_1, a_0, |\}$
Tree pattern $pref\_ranked\_bar(p_2) = a_2\ S \mid a_1\ S \mid \mid \mid$
Length of the shift:

- cannot exceed the size of the pattern, subtrees in place of S variables are expected to be smallest possible.
- is limited by the first ocurence of the particular symbol from the end. Again subtrees in place of S variables are expected to be smallest possible.
- is limited by the possible ocurence of the particular symbol in the subtree in place of the last S variable.

Bad character shift

- Again, length of the maximal safe shift is stored for each symbol.
- Both bar and ranked alphabet provide some usefull information – combination of both can be used.

Alphabet $\mathcal{A} = \{a_3, a_2, a_1, a_0, |\}$
Tree pattern $pref\_ranked\_bar(p_2) = a_2\ S\ |\ a_1\ S\ |\ |\ |$
Length of the shift:

- cannot exceed the size of the pattern, subtrees in place of S variables are expected to be smallest possible.
- is limited by the first ocurence of the particular symbol from the end. Again subtrees in place of S variables are expected to be smallest possible.
- is limited by the possible ocurence of the particular symbol in the subtree in place of the last S variable.

Bad character shift

- Again, length of the maximal safe shift is stored for each symbol.
- Both bar and ranked alphabet provide some usefull information – combination of both can be used.

Alphabet $\mathcal{A} = \{a_3, a_2, a_1, a_0, |\}$
Tree pattern $pref\_ranked\_bar(p_2) = a_2\ S\ |\ a_1\ S\ |\ |\ |$
Length of the shift:

- cannot exceed the size of the pattern, subtrees in place of S variables are expected to be smallest possible.
- is limited by the first ocurence of the particular symbol from the end. Again subtrees in place of S variables are expected to be smallest possible.
- is limited by the possible ocurence of the particular symbol in the subtree in place of the last S variable.

Bad character shift

- Again, length of the maximal safe shift is stored for each symbol.
- Both bar and ranked alphabet provide some usefull information – combination of both can be used.

Alphabet $\mathcal{A} = \{a_3, a_2, a_1, a_0, |\}$
Tree pattern $\mathit{pref\_ranked\_bar}(p_2) = a_2\ S \mid a_1\ S \mid \mid \mid$
Length of the shift:

- cannot exceed the size of the pattern, subtrees in place of S variables are expected to be smallest possible.
- is limited by the first ocurence of the particular symbol from the end. Again subtrees in place of S variables are expected to be smallest possible.
- is limited by the possible ocurence of the particular symbol in the subtree in place of the last S variable.

Bad character shift cont.

Tree pattern $pref\_ranked\_bar(p_1) = a_2\ S \mid a_1\ S \mid \mid \mid$.

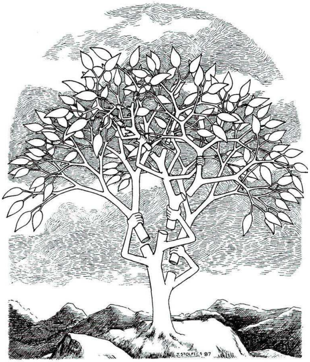
Table: Bad character shift table

	a_3	a_2	a_1	a_0	$\mid$
pattern length	8	8	8	8	8
first from right		7	4		1
inner subtree	9	7	5	3	3
min	8	7	4	3	1

a3: $a_2\ S \mid a_1\ (a_3\ a_0 \mid a_0 \mid a_0 \mid) \mid \mid \mid$
a2: $a_2\ S \mid a_1\ (a_2\ a_0 \mid a_0 \mid) \mid \mid \mid$
a1: $a_2\ S \mid a_1\ (a_1\ a_0 \mid) \mid \mid \mid$
a0: $a_2\ S \mid a_1\ (a_0) \mid \mid \mid$

Summary

- Future work
  - See if other methods of backward matching can be used for matching tree patterns.
  - Investigate if backward matching can be modified for nonlinear backward tree pattern matching.



More information on web pages

`http://www.arbology.org`

Thank you for your attention. Questions...?